U.S. ARMY RDECOM

TECHNICAL REPORT RDMR-WD-13-11

PERFORMANCE EVALUATION OF SYNTHETIC BENCHMARKS AND IMAGE PROCESSING (IP) KERNELS ON INTEL AND POWERPC PROCESSORS

Patrick A. La Fratta
Weapons Development and Integration Directorate
Aviation and Missile Research, Development,
and Engineering Center

August 2013

AMRDEC

DESTRUCTION NOTICE

FOR CLASSIFIED DOCUMENTS, FOLLOW THE PROCEDURES IN DoD 5200.22-M, INDUSTRIAL SECURITY MANUAL, SECTION II-19 OR DoD 5200.1-R, INFORMATION SECURITY PROGRAM REGULATION, CHAPTER IX. FOR UNCLASSIFIED, LIMITED DOCUMENTS, DESTROY BY ANY METHOD THAT WILL PREVENT DISCLOSURE OF CONTENTS OR RECONSTRUCTION OF THE DOCUMENT.

DISCLAIMER

THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

TRADE NAMES

USE OF TRADE NAMES OR MANUFACTURERS IN THIS REPORT DOES NOT CONSTITUTE AN OFFICIAL ENDORSEMENT OR APPROVAL OF THE USE OF SUCH COMMERCIAL HARDWARE OR SOFTWARE.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 2013	Final

4. TITLE AND SUBTITLE
Performance Evaluation of Synthetic Benchmarks and Image Processing (IP) Kernels on Intel and PowerPC Processors

5. FUNDING NUMBERS

6. **AUTHOR**(S)

Patrick A. La Fratta

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Commander, U.S. Army Research, Development, and Engineering Command
ATTN: RDMR-WDG-C
Redstone Arsenal, AL 35898-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
TR-RDMR-WD-13-11

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT ***(Maximum 200 Words)***

This report presents an in-depth performance characterization of a variety of processors released over the last decade. The processors considered include Intel and PowerPC and vary widely with respect to architectural design parameters. The benchmark experiments utilize applications from two different classes of codes. The first class, consisting of synthetic benchmarks, includes the popular Dhrystone and Whetstone suites. The second class includes a set of widely used Image Processing (IP) kernels. Following the presentation of the results from these experiments, a set of techniques for performance prediction is given based on linear correlation. This report provides an evaluation of the effectiveness of these techniques. The results show that when processors are categorized by microarchitectural families and certain restrictions to input size are employed, linear correlation shows promise for being an effective performance predictor for the IP kernels.

14. SUBJECT TERMS
Processor Benchmarking, Performance Prediction Models, Architectural Tradeoffs, Image Processing (IP), Synthetic Benchmarks, Dhrystone, Whetstone

15. NUMBER OF PAGES
38

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

NSN 7540-**01**-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

I. INTRODUCTION

Performance evaluation of processors continues to be an area of interest for military applications for a variety of reasons. Processors are ubiquitous in military systems for large scale systems (for example, the Joint Land Attack Cruise Missile Defense Elevated Netted Sensor System) and small scale systems (for example, the Integrated Hostile Fire Detection System). Tracking trends in processor performance will provide designers of such systems with valuable knowledge regarding which processors will meet performance requirements as well as improved estimates of the performance increase of future generations of processors. In addition, such performance studies will offer insights into which architectural features are most valuable for providing improvements.

In addition to the large number of processors available as options for military systems, the number of classes of applications is enormous. Different processor architectures will offer varied levels of performance for different application classes, depending on a variety of factors. Cache size, number of on-chip cores, and multi-threading support are a few of the design considerations that can have a drastic effect on how well a processor runs a particular code. For this reason, it is important that processor performance studies take into account which applications will be run on the final system.

The purpose of this work is two-fold. The first objective is to illustrate and analyze trends over the last decade in the performance of processors, including the Intel and PowerPC families, when running a variety of codes relevant to military applications. Toward this end, extensive results are provided from many processors running both synthetic and Image Processing (IP) benchmarks. The second objective is to illustrate the effectiveness of linear correlation when estimating the performance of the IP codes for various processor families.

II. BACKGROUND

A large amount of research has been completed in processor performance analysis and benchmark characterization. Among the more comprehensive studies in the area of benchmark characterization that have been published recently is a project by Demme and Sethumadhava [1]. In this work, the authors present a methodology for gauging code similarity by comparing basic blocks within functions. A novelty of this work is that the characteristics they define that are used for comparison allow for an estimation on a continuous scale of the similarity of codes, as opposed to solely a determination of whether codes are identical as provided in previous work. Other studies that are more directly related to this one have considered the question of whether correlation exists between the performances of different benchmarks on the same processor. Three studies have used Dhrystone in their experiments, which is of interest since this benchmark was used in this report. Kainaga et al. provide data and analyses to support the claim that a linear relationship exists between Dhrystone and the Standard Performance Evaluation Corporation on the processors studied [2]. However, they offer few details on the systems studied and limited discussion to support their conclusion. Munafo also makes a case for a linear relationship between Dhrystone and the Standard Performance Evaluation Corporation performance by using the results from many processors and a simple calculation from the geometric mean [3]. Aburto uses linear correlation to investigate a linear relationship between the two benchmark suites [4], which is related to the approach in this report since linear

correlation was employed. However, the key differences are that IP codes are considered rather than only standard benchmark suites. Also, processors are grouped by architectural design prior to performing the comparisons, and the processors compared are much newer.

III. METHODOLOGY

The methodology in this report consists of three subsections: an overview of the work, including the approach taken to run the benchmarks and gather data; an overview of the synthetic benchmarks and IP codes used for the performance analysis; and characterizations of the various processors considered.

A. Overview

The overall software benchmarking process was as follows:

- Source code for the benchmark was obtained (IP benchmarks were defined and written). The exception was the Fast Fourier Transform (FFT) benchmark. Its available functions had previously been compiled for a different project and was linked in as a Windows Dynamic Link Library (DLL).
- The code was compiled for the target machine(s). For example, for Windows machines, an .exe file was created.
- For the IP benchmarks, an image file consisting of random numbers was created. This was the standard image file used to input all IP benchmarks. It consisted of 14-bit unsigned integers and contained 640 columns and 512 rows (327,680 pixels). This corresponds to the maximum image size envisioned for the next generation of two-color Infrared (IR) sensors.
- For all of the convolution benchmarks, a 3-by-3 pixel kernel file was created.
- Each benchmark executable was emailed to participants, (along with the standard image and kernel files for the IP and convolution benchmarks) and any other necessary files (Windows occasionally required DLLs which were not available on all Windows machines).
- Each participant ran the benchmark. Each benchmark created a data file which contained the timing results of the run.
- The participant emailed the data file and the particulars of the machine (that is, Central Processing Unit (CPU) model number, clock speed, operating system version, Random Access Memory (RAM) size) to the benchmark coordinator who then compiled the statistics for that benchmark.

Therefore, benchmark statistics were collected on machines ranging from approximately 10 years to 6 months in age.

B. Benchmarks

This section details the software benchmarks which were obtained or written in the C programming language. These were executed on a number of available machines that were equipped with a variety of CPUs running either Windows or Linux. The compilers included Microsoft Visual Studio (Windows machines) or GNU Compiler Collection (Linux machines). Two classes of benchmarks were executed:

- Classic benchmarks consisted of Dhrystone 2.1 and Whetstone benchmarks. These were undertaken so that benchmark results quoted by Commercial Off-The-Shelf (COTS) board manufacturers could be compared to the processors accessible for this study. Another point of interest was to investigate whether this information would provide a mechanism which could estimate the execution times of these codes on other processors.

- IP benchmarks were created by the team to measure CPU performance using actual IP algorithms.

1. Synthetic Benchmarks

The classic software benchmarks were synthetic benchmarks chosen after examining a number of manufacturer's data sheets for board and chip level products. A synthetic benchmark is a program that, when run on a processing system, provides a score that is an estimate of the performance of some specific set of applications when run on that same system. Two examples of synthetic benchmarks are the Whetstone [5] and Dhrystone [6] benchmarks. The Whetstone benchmark is intended to be used for estimating the performance of numeric-scientific applications, while the Dhrystone benchmark is designed for the estimation of systems' programs. The two benchmarks have traditionally been widely used for performance evaluation of processing systems. These benchmarks were obtained from Roy Longbottom's website [7] and were used to provide some comparative metrics for the systems considered.

This following section provides an overview of the theory and design of the Whetstone and Dhrystone benchmarks. For each benchmark, there is a description of its purpose, code structure, metrics produced, how the code structure and metric were chosen, caveats, and results from the runs.

a. Whetstones

The intent behind the design of the Whetstone benchmark was to provide an estimate of the performance of numeric-scientific programs. Numeric-scientific programs are compute-intensive codes containing a variety of operation types, such as integer, floating point, trigonometric, and others. The designers of the Whetstone benchmark sampled the instruction frequencies of 949 numeric-scientific programs and used these statistics to create the code.

The Whetstone code consists of a main function that executes eight loops or modules. Each module is assigned a weight factor, which governs the number of times the module is executed and was determined using the statistics from the 949 programs sampled. The benchmark prints results reporting the performance of each module. Three of these results are in

megaflops for the floating point-intensive modules and five results are in megaops for the other modules.

The metric that is most frequently reported with the use of Whetstone is Millions of Whetstone Instructions Per Second (MWIPS). The history behind the design of the benchmark reveals that the instruction set of the sampling process was the intermediate code for the Whetstone system from the 1960s, which is how the benchmark got its name. Each module in the code is executed a certain number, n[1], of iterations. This number of iterations times ten gives the millions of Whetstone Instructions (that is, instructions for the Whetstone System) to which the code would compile. The time required to run all the modules on a system is divided into 10n to give that system's performance in MWIPS.

In Reference 5, Curnow and Wichmann give some caveats when using the benchmark to measure performance. The authors state that while the benchmark may provide a performance estimate for some machine *M* when running the 949 sample programs, it is not intended to provide an estimate of other applications when run on *M*.

"The benchmark program described here has been presented as a model of the large number of programs originally analyzed. The intention is that by running it upon a new type of machine one may learn something of the performance the machine would have if it ran the original programs" [5].

When run on machine *M*, it would be incorrect to assume that the results can be used directly to provide insight into the performance of the original codes on some other machine.

"When more is known … it may be possible to produce a typical program for particular types of machine. It will clearly be impossible to produce one valid for any conceivable machine" [5].

A primary reason for these caveats is that subtleties of the system design as a whole can result in significant differences in performance.

"It may well be true that on the 360/65 the use made by the FORTRAN H compiler of the general purpose registers was reasonably typical, although it did manage to perform the whole of module 4 in registers" [5].

Note that these specific caveats should, in general, be taken into consideration when using benchmarks to measure performance.

b. Dhrystones

The Dhrystone benchmark was designed to provide a performance estimate of set of systems programs, which are described as programs that "often use enumeration, record, and pointer data types" [6]. To gather statistics for use in the design of

[1] This is represented in the code with the variable *xtra*.

Dhrystone, the authors sampled 16 program suites, including compilers, a very large scale integration checking program, and a Computer-Aided Design (CAD) tool.

The Dhrystone code consists of a main loop that calls eight procedures, whose content was selected based on the results of the sampling of the program suites. The loop iterates *n* times, which is the number of iterations the system can execute in *t* time. The duration *t* is controlled by the code and set to approximately 2 seconds. The primary metric reported by the benchmark is Dhrystones per second, which is simply *n*/*t*. The result is often normalized to the performance of a particular Virtual Address eXtension (VAX) machine that was capable of 1757 Dhrystones per second. Hence, the VAX Millions of Instructions Per Second (MIPS) for a machine is its Dhrystones per second divided by 1,757. The VAX MIPS score is frequently referred to as Dhrystone Millions of Instructions Per Second (DMIPS). Note that this terminology is convoluted. "MIPS" mean *millions of instructions per second*, and the "instructions" within the VAX MIPS and DMIPS abbreviations don't correspond to any actual architecture (it depends on for which architecture the code is compiled). This is contrasted with MWIPS, where the instruction counts correspond to instructions for the Whetstone architecture. The name Dhrystone was chosen to allude to Whetstone, which came earlier. This point is very important when forming an intuition as to the meaning of the terms MWIPS and DMIPS and interpreting results that use these metrics.

One of the caveats when using Dhrystones is that because of the small code size, the benchmark may give an unintended advantage to processors with large caches. A processor with a cache above a certain size may be able to fit all code and data needed for the benchmark's execution in the cache, which may drastically reduce execution time.

"Dhrystone's intended ease of implementation, however, has consequences (e.g., cache influences) that must be taken into account if the program is to be used to compare different computer architectures or different compilers" [6].

Multiple times Weicker emphasizes that using any benchmark to measure performance must be done with great care, as there are many subtle factors in system design and application characteristics that can influence results.

"…there are inherent limitations to any single number (like a benchmark result) if it is used as the *only* criterion for the evaluation of processor architectures…" [6].

2. IP

The IP software benchmarks were created to measure execution times of real-world IP codes. This has several advantages (as compared to extrapolation from classic benchmarking algorithms) including giving actual execution times (as opposed to an operations per second rating), accounting for overhead operations typical of IP operations (for example, computation of indices), and the use of large data sets (as compared to the classic benchmarks) typical of images. The particular algorithms chosen are typical IP algorithms. Each benchmark executes its particular algorithm against varying numbers of the pixels in the standard image file. In that way, any data size dependencies (especially those that might be associated with limited amounts of cache) might be discovered along with an indication of where (at what data size) they

occur. The algorithms were also executed using different numeric representations on available machines. The algorithms chosen are summarized in Table 1, along with the numeric representations used for each algorithm in Table 2 and the machines used in Table 3.

Table 1. IP Algorithms Chosen

Algorithm	Equation	Typical Use
Brightness Adjust	$O(x,y) = I(x,y) + K$	Lighten or darken image.
Contrast Adjust	$O(x,y) = K * I(x,y)$	Adjust image dynamic range.
Image Difference	$O(x,y) = I_1(x,y) - I_2(x,y)$	Moving Target Indication (MTI) or detection.
Pixel Threshold	$O(x,y) = \begin{cases} 1 \ if \ I(x,y) > K \\ 0 \ otherwise \end{cases}$	Detection of light or dark objects.
Image Ratio	$O(x,y) = I_1(x,y) / I_2(x,y)$	Reducing clutter for two-color IR sensing systems.
Convolution	$O(x,y) = \sum_{k=0}^{2}\sum_{j=0}^{2} I_1(k,j) * I_2(x-k+2, y-j+2)$	Filtering of images, pattern matching, and correlation processing.
FFT	See the paper "fftw3.pdf" at http://www.fftw.org/fftw3.pdf.	Conversion from time/spatial to frequency domains, typically prior to applying complex filters.
Conversions	N/A	Conversions from integer data type to float and double data types.

Note: O(x, y) is an output pixel, I1(x,y) is an input pixel from Image 1 (or a kernel pixel for Convolution), I2(x,y) is an input pixel from Image 2, and K is a constant.

Table 2. Data Types Used for Each Algorithm in IP Benchmarks

	Data Type			
Algorithm	**Integer (32 bit)**	**Fixed Point**	**Float (32 bit)**	**Double (64 bit)**
Brightness Adjust	X	X		
Contrast Adjust	X	X	X	
Image Difference	X	X	X	
Pixel Threshold	X	X	X	
Image Ratio				
Convolution	X	X	X	X
FFT			X	X
Conversions (from integer to …)			X	X

Table 3. Summary of Classic Benchmark Data

Processor	Clock Speed (GHz)	# Cores/ Threads	Whets Int (unoptimized)	Whets Float (unoptimized)	Dhrystone DMIPS (optimized)	Operating System	Release Date
Pentium 4	1.8	1/1	491	379	1774	Linux	Q1 2002
Pentium 4 (520)	2.8	1/1	1571	1187	2949	Linux	Q1 2002
Pentium 4 (550)	3.4	1/2	3439	1304	3734	Windows XP	Q2 2004
Cell Broadband Engine	3.2	1/1	239	441	2006	Linux	Q1 2005
Pentium D (830)	3	2/2	2511	1148	3617	Windows Vista	Q2 2005
Pentium D (830)	3	2/2	2938	1155	3556	Windows XP	Q2 2005
PowerPC 970MP	2	2/2	375	942	4396	Linux	Q3 2005
AMD Athlon Dual Core 4400	2.2	2/2	2699	1772	3869	Windows 7	Q3 2005
AMD Athlon Dual Core 4400	2.2	2/2	1979	1833	6834	Linux	Q3 2005
Pentium D (950)	3.4	2/2	2898	1307	3872	Windows 7	Q1 2006
Core 2 Quad 6600	2.4	4/4	3044	1724	4050	Windows 7	Q1 2007
Core 2 Duo E6550	2.33	2/2	4188	2548	5710	Windows XP	Q3 2007
Core 2 Duo E6750	2.66	2/2	5124	2911	6543	Windows XP	Q3 2007
Core 2 Duo E6750	2.66	2/2	4376	2908	6863	Windows XP	Q3 2007
Core 2 Duo E8500	3.16	2/2	5385	3454	7479	Windows Vista	Q1 2008
Atom N270	1.6	1/2	755	653	2217	Linux	Q2 2008
Core 2 Duo P8400	2.26	2/2	4267	2611	6092	Windows XP	Q3 2008
Core 2 Duo T9600	2.8	2/2	5172	3125	6984	Windows 7	Q3 2008
Core 2 Duo P8700	2.53	2/2	4499	2834	6135	Windows XP	Q4 2008
Core i7 (620M)	2.67	2/4	5022	3325	7735	Windows 7	Q1 2010
Core i7 (930)	2.8	4/8	5311	3066	8389	Windows 7	Q1 2010
Core i5 (650)	3.2	2/4	5924	3465	9473	Windows 7	Q1 2010

The single-threaded benchmarks were written for execution on a single CPU core. Therefore, they could be executed on both single- and multi-core processors. For multi-core processors, they give the performance of a single core. Each algorithm is categorized as simple, moderately complex, or very complex, depending on the computational complexity of the algorithm and hence its execution time.

The results of the moderately and very complex benchmarks implied that these algorithms would need to be partitioned and executed on multi-core processors or field programmable gate arrays if real-time performance was desired. To investigate the speedup possible with multi-core processors, the simple algorithms were rewritten (the 640-by-512 image was partitioned into four evenly sized image sections) with each partition assigned to a thread. Because of a lack of time, these multi-threaded benchmarks were run only on multi-core or hyperthreaded processors using Microsoft Windows. It was assumed that Windows would assign each thread to a different core or thread processor.

a. Brightness Adjust

This algorithm takes each pixel in the input image and adds a constant, producing an output image of the same size which is brighter (or darker if the constant is negative) than the original image. Since this function is usually used as an intermediate operation, no check is made to limit pixel values. Mathematically, this function can be expressed as:

$$O(x, y) = I(x, y) + K \tag{1}$$

for all x,y in the Input Image I, where *O(x, y)* is an output pixel, *I(x,y)* is an input pixel, and *K* is a constant.

b. Contrast Adjust

This algorithm takes each pixel in the input image and multiplies it by a constant, producing an output image of the same size which has more contrast (or less contrast if the constant is less than 1) than the original image. Since this function is usually used as an intermediate operation, no check is made to limit pixel values. Mathematically, this function can be expressed as:

$$O(x, y) = K * I(x, y) \tag{2}$$

for all x,y in the Input Image I, where *O(x, y)* is an output pixel, *I(x,y)* is an input pixel, and *K* is a constant.

c. Image Difference

This algorithm takes two input images of the same size and subtracts each pixel in Input Image 2 from the corresponding pixel in Input Image 1. This produces an output image of the same size which represents the difference between the two images. This algorithm is often used for detecting changes in images created at two different times. Since this function

is usually used as an intermediate operation, no check is made to limit pixel values. Mathematically, this function can be expressed as:

$$O(x,y) = I_1(x,y) - I_2(x,y) \tag{3}$$

for all x,y in the Input Images I_1 and I_2, where $O(x, y)$ is an output pixel, $I_1(x,y)$ is an input pixel from Image 1, and $I_2(x,y)$ is an input pixel from Image 2.

d. Threshold Algorithm

This algorithm takes each pixel in the input image and compares it to a constant. If the pixel value is greater than the threshold value, the value 1 is placed in the corresponding pixel of the output image. Otherwise, a 0 is placed in the corresponding pixel of the output image. Mathematically, this function can be expressed as:

$$O(x,y) = \begin{cases} 1 \ \ if \ \ I(x,y) > K \\ 0 \ otherwise \end{cases} \tag{4}$$

for all x,y in the Input Image I, where $O(x, y)$ is an output pixel, $I(x,y)$ is an input pixel, and K is a constant.

e. Image Ratio

This algorithm takes two input images of the same size and divides each pixel in Input Image 1 by the corresponding pixel in Input Image 2. This produces an output image of the same size which represents the ratio of the pixel values. This algorithm can be used for reducing clutter in a two-color IR missile sensor. Since this function is usually used as an intermediate operation, no check is made to limit pixel values. However, to avoid division by 0, the function adds 1 to every pixel in both input images prior to performing the division. This simple method avoids the division by 0 problem without significantly affecting the resulting ratios. Mathematically, this function can be expressed as:

$$O(x,y) = I_1\ (x,y) / \ I_2(x,y) \tag{5}$$

for all x,y in the Input Images I_1 and I_2, where $O(x, y)$ is an output pixel, $I_1(x,y)$ is an input pixel from Image 1, and $I_2(x,y)$ is an input pixel from Image 2.

f. Convolution

This algorithm takes two input images (typically called input and kernel images) and convolves them. Convolution is often used for extracting features (such as edges) from an input image or in filter operations (such as a low-pass noise filter). Also, since convolution is mathematically very similar to correlation, in certain cases, it can be used as a less (computationally) expensive alternative to correlation.

The kernel was a constant 3-by-3 pixel image file, so the smallest input image is also 3-by-3 pixels. Using the 3-by-3 kernel produces an output image of size (C-2) columns x (R-2) rows, where C and R are the number of columns and rows in the input image,

respectively. Despite that this benchmark uses a fixed 3-by-3 kernel, the results are readily extensible to other sized kernels.

Mathematically, the convolution algorithm as written can be expressed as:

$$O(x,y) = \sum_{k=0}^{2}\sum_{j=0}^{2} I_1(k,j) * I_2(x-k+2,\ y-j+2) \quad (6)$$

for all x ≤ (C-2) and y ≤ (R-2) in I_2, where $O(x, y)$ is an output pixel, $I_1(k,j)$ is a kernel input pixel, and $I_2(x-k+2,y-j+2)$ is a pixel from the Input Image I_2.

g. FFT

This algorithm computes the FFT of the input image. The FFT is a basic signal processing tool that is used in a large number of algorithms. Because of the large amount of effort and extensive literature concerning efficient computation of the FFT, the team decided to use a readily available package which does the computation [8]. This package computes the FFT of input data of arbitrary size using perhaps the most efficient algorithms available. In the benchmark implementation, subimages of various sizes are extracted from the standard image and FFT is computed and execution times recorded.

h. Conversion

These algorithms measured the time necessary to convert from the 14-bit unsigned integer image input data to the float or double data types (both using Institute of Electrical and Electronics Engineers (IEEE) 754 representations) used in some algorithms. In these algorithms, each input image pixel is converted to type float or double and placed in the corresponding location in an output image using the C-type cast.

C. Processors

This work uses numerous processors for experimentation. The processors are characterized in Table 3, which shows the processors on which the classic benchmarks were run, and Table 4, which shows the processors on which the IP codes were run. These processors vary across a number of different dimensions, including release date, clock speed, and number of cores. Other variables across the processors not shown in the table include microarchitectural family and cache size. While the Instruction Set Architectures considered included PowerPC and x86, there were many more x86 machines used. The different x86 microarchitectures included NetBurst, Core, and Nehalem from Intel and the K7 from Advanced Micro Devices (AMD). In addition to the varying processor designs, the operating systems run on the processors varied, as did the compilers used to generate the codes. Table 3 summarizes the results from the classic benchmarks. Intel defines Thermal Design Power (TDP) as "the maximum power a processor can draw for a thermally significant period while running commercially useful software." Table 4 shows the TDP of each processor, which offers a rough estimate of the power consumption under normal operating conditions.

Table 4. Summary of Machines Using IP Benchmarks

Processor	Clock Speed (GHz)	# Cores/ Threads	Memory (Gbytes)	TDP (Watts)	Operating System	Launch Date	Technology (nm)	Machine Designator
Pentium 4	1.8	1/1	0.5	68.1	Li	Q1 2002	130	Ild
Pentium 4	2.8	1/2	1	84	Li	Q1 2002	90	Di3
Pentium 4 (550)	3.4	1/2	1	115	XP	Q2 2004	90	GhP4
Cell Broadband Engine	3.2	1/16	0.207	92	Li	Q1 2005	90	Ic
Pentium D (830)	3	2/2	2	130	V	Q2 2005	90	B
Pentium D (830)	3	2/2	2	130	XP	Q2 2005	90	Hgd
PowerPC 970MP	2	2/2	2	70	Li	Q3 2005	90	Dg5
AMD Athlon Dual Core 4400	2.2	2/2	3	89	W7	Q3 2005	90	Ddbw
AMD Athlon Dual Core 4400	2.2	2/2	3	89	Li	Q3 2005	90	Ddbl
Pentium D (950)	3.4	2/2	2	130	W7	Q1 2006	65	Di1
Core 2 Quad 6600	2.4	4/4	2	105	W7	Q1 2007	65	Di2
Core 2 Duo E6550	2.33	2/2	1.96	65	XP	Q3 2007	65	F
Core 2 Duo E6750	2.66	2/2	3.25	65	XP	Q3 2007	65	Cd
Core 2 Duo E6750	2.66	2/2	3.25	65	XP	Q3 2007	65	Gg
Core 2 Duo E8500	3.16	2/2	4	65	V	Q1 2008	45	J1
Atom N270	1.6	1/2	1	2.5	Li	Q2 2008	45	Da
Core 2 Duo P8400	2.26	2/2	2	25	XP	Q3 2008	45	A
Core 2 Duo T9600	2.8	2/2	8	35	W7	Q3 2008	45	Ghl
Core 2 Duo P8700	2.53	2/2	2	25	XP	Q4 2008	45	M
Core i7 (620M)	2.67	2/4	8	35	W7	Q1 2010	32	Iw
Core i7 (930)	2.8	4/8	9	130	W7	Q1 2010	45	Ghi7
Core i5 (650)	3.2	2/4	4	73	W7	Q1 2010	32	J2

IV. PERFORMANCE RESULTS

This section presents the results from the experiments along with discussions of these results. The first section gives consideration to the results from the synthetic benchmarks in both graphical and tabular form, while the second section gives the results from the IP codes. Each section gives a discussion of important trends in these results, followed by a summary of the conclusions drawn from them.

A. Synthetic Benchmarks

Figure 1 shows a graphical summary of the results from the synthetic benchmarks with the processors ordered by release date. Certain characteristics, such as processor family and clock rate, are highlighted for various processors to facilitate easier recognition of trends in the graph. Table 5 shows the results from these experiments but with further detail for each of the processors used, including clock speed, number of cores, number of threads, and total memory.

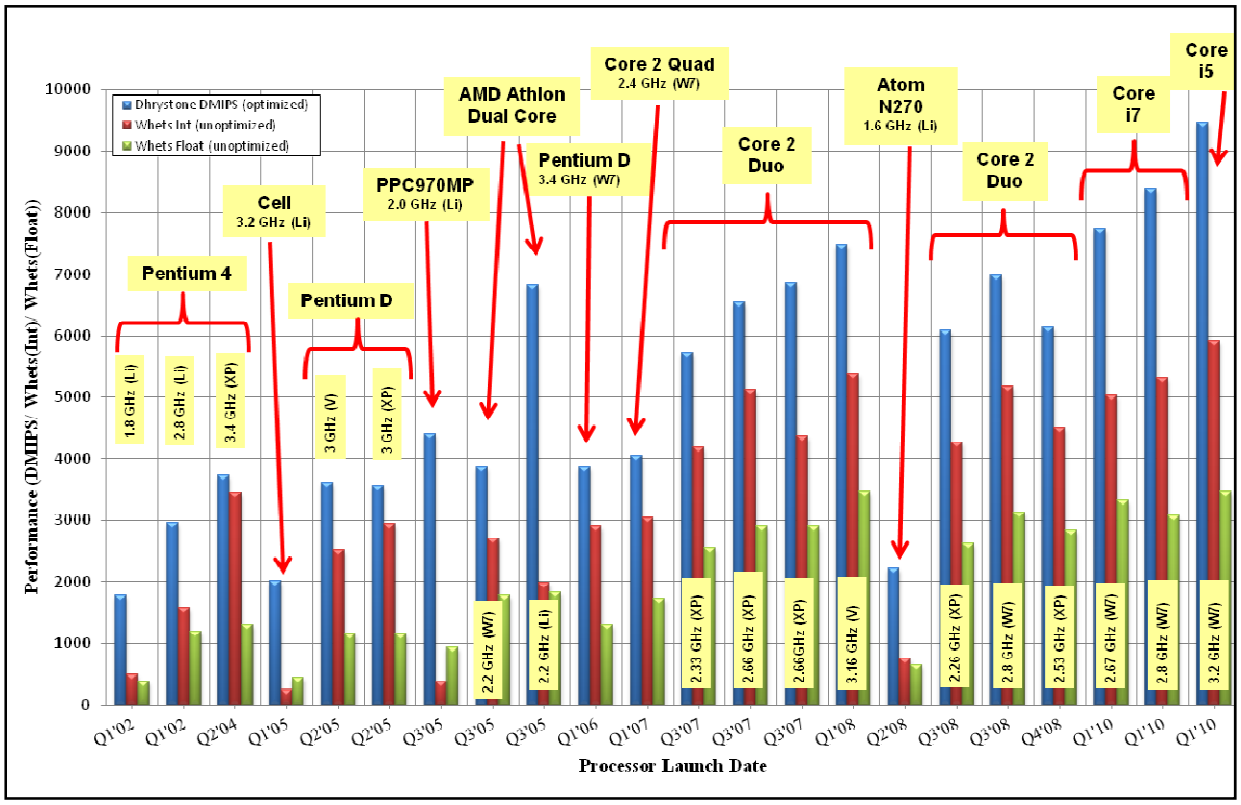

Figure 1. Classic Benchmark Results

Table 5. Classic Benchmark Results (Larger Numbers Indicate Better Performance)

				Measured Data		
Processor	**Clock Speed (GHz)**	**# Cores/ Threads**	**Memory (Gbytes)**	**Dhrystone DMIPS (optimized)**	**Whets Int (unoptimized)**	**Whets Float (unoptimized)**
Pentium 4	1.8	1/1	0.5	1774	491	379
Pentium 4	2.8	1/2	1	2949	1571	1187
Pentium 4 (550)	3.4	1/2	1	3734	3439	1304
Cell Broadband Engine	3.2	1/1	0.207	2006	239	441
Pentium D (830)	3	2/2	2	3617	2511	1148
Pentium D (830)	3	2/2	2	3556	2938	1155
PowerPC 970MP	2	2/2	2	4396	375	942
AMD Athlon Dual Core 4400	2.2	2/2	3	3869	2699	1772
AMD Athlon Dual Core 4400	2.2	2/2	3	6834	1979	1833
Pentium D (950)	3.4	2/2	2	3872	2898	1307
Core 2 Quad 6600	2.4	4/4	2	4050	3044	1724
Core 2 Duo E6550	2.33	2/2	1.96	5710	4188	2548
Core 2 Duo E6750	2.66	2/2	3.25	6543	5124	2911
Core 2 Duo E6750	2.66	2/2	3.25	6863	4376	2908
Core 2 Duo E8500	3.16	2/2	4	7479	5385	3454
Atom N270	1.6	1/2	1	2217	755	653
Core 2 Duo P8400	2.26	2/2	2	6092	4267	2611
Core 2 Duo T9600	2.8	2/2	8	6984	5172	3125
Core 2 Duo P8700	2.53	2/2	2	6135	4499	2834
Core i7 (620M)	2.67	2/4	8	7735	5022	3325
Core i7 (930)	2.8	4/8	9	8389	5311	3066
Core i5 (650)	3.2	2/4	4	9473	5924	3465

The first noteworthy point about the results is that although there is a general trend of increasing performance with release date, there are clear exceptions to this trend. While some exceptions have straightforward explanations, the reasons for these exceptions may not be immediately obvious. An example of a simple explanation is the Atom N270, which was released in the second quarter of 2008 but showed a lower performance than the Pentium 4 that was released more than 6 years earlier. The Atom family of processors was designed for mobile devices with low power as a primary design constraint. In general, processors trade power for performance. The Atom has the lowest TDP by a large margin among the processors considered. Its TDP is 2.5 watts, and the next lowest power processors (the Core 2 Duo P8400 and the Core 2 Duo P8700) have TDPs of 10 times that. Note that performance does not always increase with TDP, even within a processor family. The Core i5 has a TDP of 73 watts but outperforms the Core i7 that has a TDP of 130 watts on all classic benchmarks although both are Nehalem-based processors. Other cases are more difficult to explain, such as the relative performance of the Pentium D 830 and the Pentium 4 550. The Pentium D 830 has a higher TDP and is newer than the Pentium 4 550, but the Pentium 4 550 outperformed it on all benchmarks. Note that a higher clock rate does not always translate into better performances across processor families. The data show that performance generally follows clock rate, but there are exceptions such as the Core 2 Duo E6550 running at 2.33 gigahertz and the Core 2 Duo P8400 running at 2.26 gigahertz, with the latter outperforming the former on all benchmarks.

Another point worth considering is that while the data reflect that multi-core processors have become more prominent in recent years, performance does not always improve with increased cores. The Core 2 Quad 6600 is a processor with four cores but is outperformed on all benchmarks by all of the dual-core processors in subsequent years. It is even outperformed by one of the single-core processors on one benchmark from a previous year (the Pentium 4 550 running Integer Whets). These benchmarks were not implemented to utilize multiple cores, and since the team ran them with negligible loads on other cores (other than the core running the benchmark itself)[2], performance was primarily a function of design parameters rather than the number of cores (such as design of individual cores, cache size, cache controller design, and so forth).

This previous point explains, to some extent, why multiple cores do not necessarily translate into higher performance. However, there is a general trend of increasing performance with later release dates which warrants further consideration. Note that the newer processors tend to have larger caches. The oldest processor considered (a Pentium 4) has at most a 256 kilobyte L2 cache, while the newer Core i7 930 has an 8 megabyte L3 cache. Although this may be a first consideration for explaining performance increases for data-intensive applications, Whetstones and Dhrystones are relatively small codes with small working sets that can most likely fit into the older processors' caches. However, the speed of these caches and other components outside of the processing logic could definitely come into play because the codes cause activity in these components, such as during compulsory cache misses that are included in the timing data. Other components that have likely improved with newer processors that could have an effect on performance are branch predictor accuracy, number of functional units per

[2] The operating system is running in the background, and its utilization of the processor can vary widely. It is assumed that the load that the operating system places on the processor is negligible, although this may be an inaccurate assumption for more in-depth discussions.

core, and instruction scheduling hardware and configuration. A route for getting a clearer answer on what are the biggest factors which are effecting performance within and across processor families would be profiling and simulation-based analyses.

A final point is that there are other factors that are independent of the processor that have effects on performance. Other factors that may come into play are the operating systems, libraries, run-time instrumentation, and software build options. To consider the first of these, the benchmarks were run on a machine (the AMD Athlon Dual Core 4400) with both Windows 7 and Ubuntu Linux installed. The results show that even though the benchmarks were run on the same processor, performance under different operating systems varied. The use of Linux over Windows 7 resulted in a significant increase in performance for Dhrystone (a 77-percent improvement). On the other hand, Windows 7 outperformed Linux on Integer Whets (a 36-percent improvement). They performed about the same on Float Whets. Note that different compilers were used for different operating systems (Visual Studio for Windows and GNU Complier Collection for Linux), so further study would be needed to determine which factors contributed primarily to the disparity in performance. These results reveal that other components besides the processor and application can have significant effects on performance.

To summarize the important observations from this data:

- The overall trend indicates newer processors have larger ratings (are faster).

- In most cases, the ratings appear to be correlated. When comparing two processors, if processor A has a higher Dhrystone rating than processor B, then usually the Whetstone ratings for processor A will also be higher than that of processor B. This is especially true for newer processors whose launch date is the first quarter of 2007 and later. However, there are exceptions to this. The Core i7 620M has a higher Whet Float rating than the Core i7 930, but the Core i7 930 outperformed the Core i7 620M on Dhrystones.

- When comparing the performance of newer processors (launch date is the first quarter of 2007 and later), it appears that Dhrystone performance has continued to improve with time, but Whetstone performance has improved at a slower rate.

- The AMD Athlon Dual Core processor is a dual-boot machine whose benchmarks were compiled under Linux (GNU compiler) and Microsoft Visual Studio version 8. The results give an interesting view into the role of the compiler and/or operating system in use. The Float Whets for both are almost identical, but the integer performance is significantly higher (2,699 versus 1,979) for the Visual Studio 8 compilation run under Windows 7. However, the Dhrystone performance is significantly higher (6,834 versus 3,869) for the GNU compiler run under Linux.

B. Image Processing Kernels

The IP kernels are split into three categories: simple, moderately complex, and very complex. The simple kernels in general consist of a simple loop containing a primitive operation that takes as input pixels from one or two images. The moderately complex codes consist of either a more computationally expensive operation (such as ratio), a more complex data type (such as single-precision floating point numbers), or an algorithm of higher order complexity (such as FFT). The very complex kernels include the FFT with double-precision floating point, along with other codes implementing higher-complexity algorithms. The simple codes were implemented with multi-threading.

The following graphs plot execution time. Note that a lower value indicates better performance. The overall trends in these results look in some ways similar to those for the synthetic benchmarks, which suggest that there may be correlations between the two. We consider this question in detail in the next section. As with the synthetic kernels, it is difficult to make any generalizations about correlations between performance and processor characteristics. For example, it is clear that neither clock rate nor number of cores imply better performance. Also, the results suggest that other factors, such as operating system or compiler used, can have a non-negligible impact on performance.

1. Single-Threaded "Simple" Results

Several interesting trends are shown in Figure 2:

- Better performance (lower execution time) is available from newer processors, despite that clock rates for the newer processors are not generally higher than many older processors. Remember that these numbers are for single-threaded benchmarks and therefore do not take advantage of multi-core or multi-threaded CPUs. It is likely that the newer processors (having more transistors according to Moore's law) have architectural improvements that translate into better performance.

- Within a particular family, faster clock rates generally—but not always—translate into better performance.

- Again, processors designed for low power applications (for example, the Pentium M and Atom processors designed specifically for laptops) sacrifice computing performance to achieve their low power attributes.

- The cell processor is an interesting combination of a 64-bit PowerPC core with eight Synergistic Processing Elements (SPEs) [10]. Because of the single-threaded nature of the benchmarks, the SPEs are inactive. The poor performance of the PowerPC core within the cell implies that the designers used the available chip area (that is, available transistors) to implement the SPEs as opposed to using them for architectural enhancements to the PowerPC core.

- For most of these simple algorithms, most of the newer processor families (Core 2, Core i5, and Core i7) can execute the algorithms in under 0.5 milliseconds. This implies essentially real-time processing of a video frame (at the given 2.0 millisecond frame time) with less than a 50-percent processor loading.

- Floating point performance of most of the processors rivals that of integer and fixed point operations.

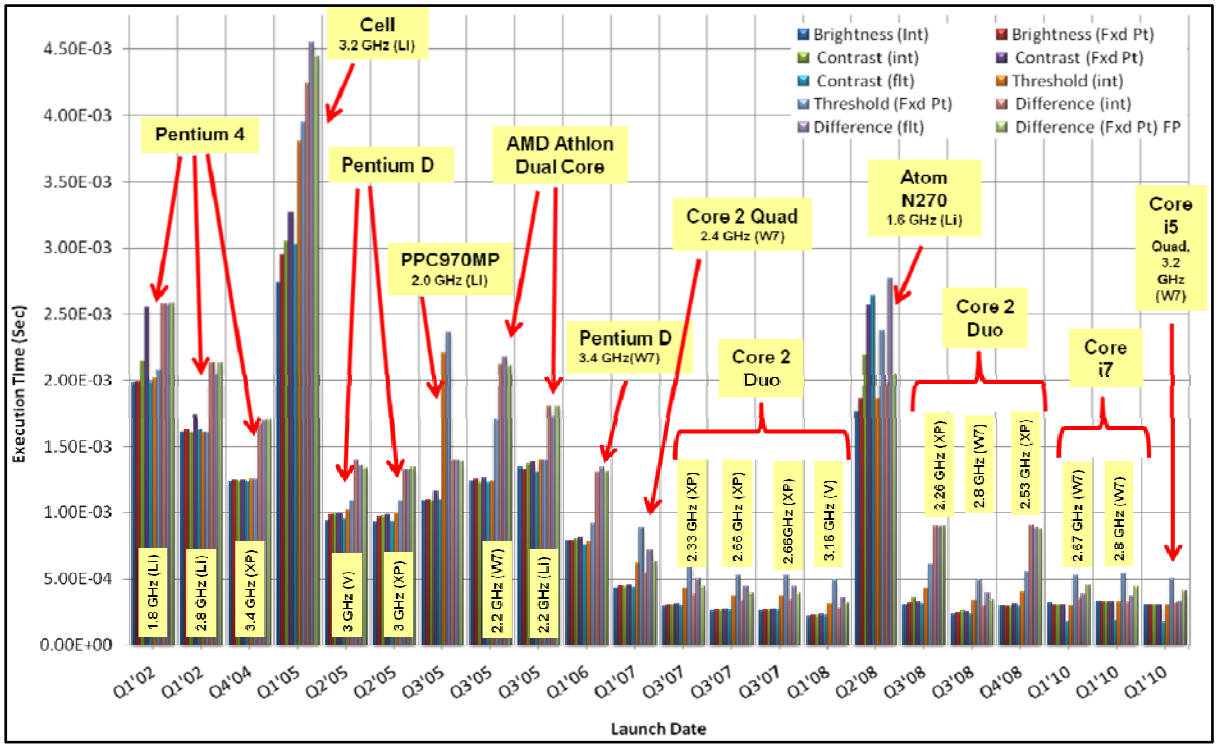

Figure 2. Results for Single-Threaded "Simple" Benchmarks

2. Single-Threaded "Moderately Complex" Results

Figure 3 shows a plot of execution times for various "moderately complex," single-threaded benchmarks, which includes information on the processors and their release dates. Many of the previously noted trends are apparent but with the following important exception: algorithms of this complexity will generally not execute within the image frame time (2.0 milliseconds) on a single core. However, recently released dual- and quad-core processors may be able to execute them in time if the algorithm can be efficiently partitioned among the cores. This was the impetus behind the creation of the multi-threaded benchmarks discussed in following sections of this report.

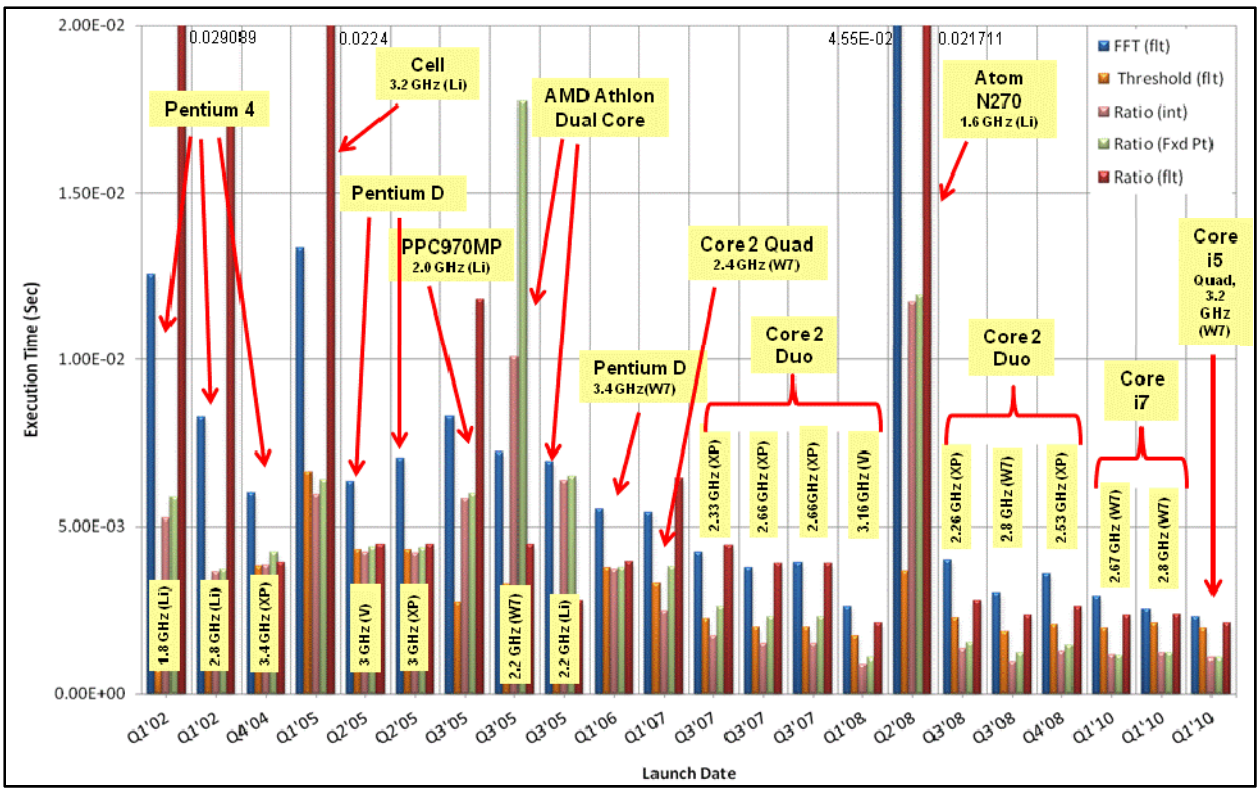

Figure 3. Results for Single-Threaded "Moderately Complex" Benchmarks

3. Single-Threaded "Very Complex" Results

Figure 4 shows a plot of execution times for various "very complex" single-threaded benchmarks, including information on processors and their release dates. Many of the trends previously noted are again apparent, but it is clear that algorithms of this complexity will certainly not execute within the image frame time (2.0 milliseconds) on a single core. Execution of these algorithms in real time will require the full resources of a quad-core processor or a specialized processing engine such as a field programmable gate array.

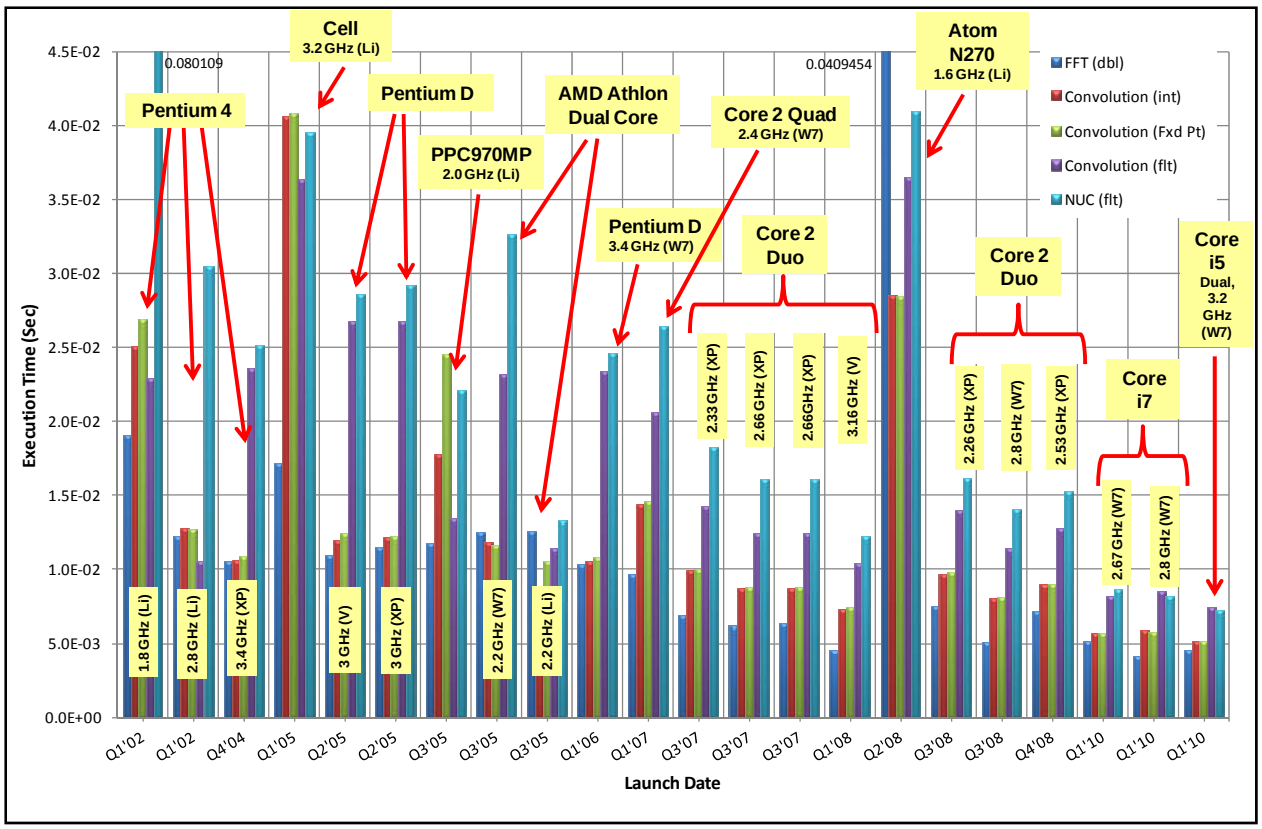

Figure 4. Results for Single-Threaded "Very Complex" Benchmarks

4. Multi-Threaded Results

Figure 5 shows the results from the multi-threaded implementations of the simple benchmarks, and it gives both the average performance for the single-threaded implementation and multi-threaded implementations of the codes to offer an easy comparison. Clearly, the optimal speedup was not often obtained. One possible explanation was that this was an artifact of the Windows scheduling algorithm. These results make clear that careful algorithm partitioning and control over thread-to-processor assignments are critical to achieve speedups close to the theoretical maximums. Balancing the per-processor load is not trivial and will require a talented and knowledgeable software team.

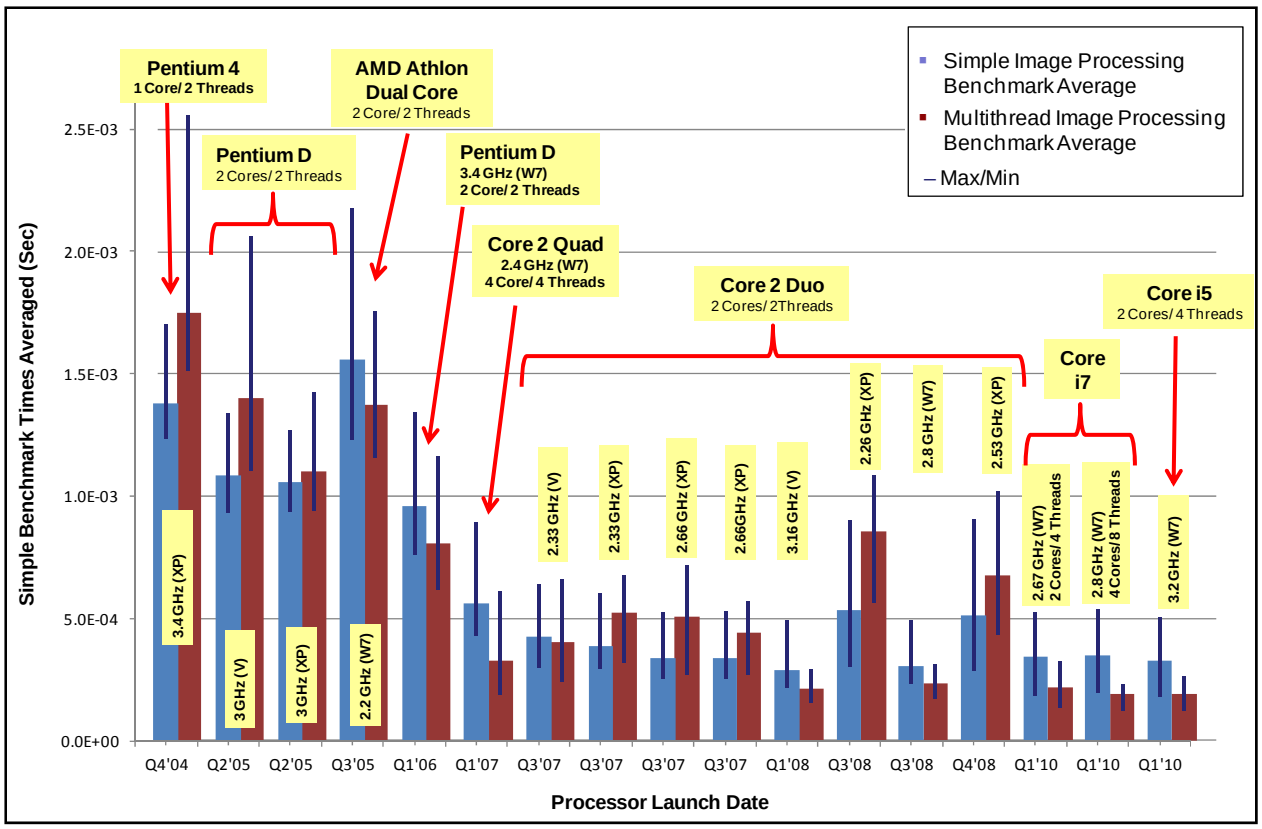

Figure 5. Results for Multi-Threaded Benchmarks

V. PREDICTING IMAGE PROCESSING PERFORMANCE WITH SYNTHETIC RESULTS USING LINEAR REGRESSION

This section disscusses performance results from the various applications and provides an in-depth analysis of the data to answer questions regarding the estimation of the performance of the IP codes. Graphs showing the results for the IP codes versus image size form the basis for the discussions. A model of the performance of the processors within a given family when running the IP codes through multiple regression is given. The conclusions consider the question of whether there are relationships between trends in the performance of the IP applications with respect to this regression model and the synthetic benchmarks.

This section considers whether the IP code execution time on a particular processor could be expressed as a function of four variables:

- The execution time of the Whetstone for the processor
- The execution time of Dhrystone for the processor
- The number of pixels in the image being processed
- The family in which the processor falls[3]

An equation is generated for each processor family for estimating the execution time of IP codes. Multiple regression was used with Whetstone execution time, Dhrystone execution time,

[3] "Family" refers to the processor's microarchitectural family.

and number of image pixels as independent variables to generate a linear approximation for each processor family for various IP codes. In the following equations, the variable t_{x_y} represents an estimate of the execution time (in seconds) for application *x* when run on a processor in family *y*. The codes considered are brightness with integer data (*BI*); contrast with fixed point data (*CF*); and threshold with single-precision, floating-point data (*TH*). The processors considered are from three families of Intel processors: NetBurst, Core, and Nehalem. The *W* is the average time (in milliseconds) per loop for the Whetstone benchmark for the processor, the *D* is the average time (in milliseconds) per loop for the processor, and the *p* is the number of pixels in the image being processed[4].

$$t_{BI\_NetBurst} = -1.729x10^{-3} + 5.601x10^{-5} * W + 4.429 * D + 3.320x10^{-9} * p \quad (7)$$

$$t_{BI\_Core} = -1.210x10^{-4} + 2.715x10^{-5} * W - 8.470x10^{-1} * D + 9.115x10^{-10} * p \quad (8)$$

$$t_{BI\_Nehalem} = -1.620x10^{-4} + 3.973x10^{-5} * W - 1.065 * D - 1.096x10^{-9} * p \quad (9)$$

$$t_{CF\_NetBurt} = -1.503x10^{-3} + 4.736x10^{-5} * W + 3.831 * D + 3.434x10^{-9} * p \quad (10)$$

$$t_{CF\_Core} = -1.402x10^{-4} + 4.093x10^{-5} * W - 1.469 * D + 9.865x10^{-10} * p \quad (11)$$

$$t_{CF\_Nehalem} = -1.468x10^{-4} + 4.313x10^{-5} * W - 1.442 * D + 1.086x10^{-9} * p \quad (12)$$

$$t_{TH\_NetBurst} = -1.577x10^{-3} + 6.449x10^{-5} * W + 2.732x10^{-1} * D + 1.400x10^{-8} * p \quad (13)$$

$$t_{TH\_Core} = -8.662x10^{-4} + 1.628x10^{-4} * W - 4.536 * D + 7.101x10^{-9} * p \quad (14)$$

$$t_{TH\_Nehalem} = -2.38010^{-4} + 2.689x10^{-5} * W + 1.634x10^{-1} * D + 1.830x10^{-9} * p \quad (15)$$

Figures 6 though 14 show the execution time of each of the processors for the different IP codes versus image size in number of pixels. The legend for each chart shows the average time per loop for Whetstone and Dhrystone for each processor. The graphs also show lines generated from the previous equations for each processor. These lines represent approximations of the actual performance curves. Note that only a portion of these lines are shown to make the graphs more readable because it is straightforward to mentally extrapolate the lines back to the Y-axis and out to the right edge of the graph.

[4] The time per loop for the Whetstone and Dhrystone benchmarks can be calculated from the scores from each benchmark and is inversely proportional to these scores.

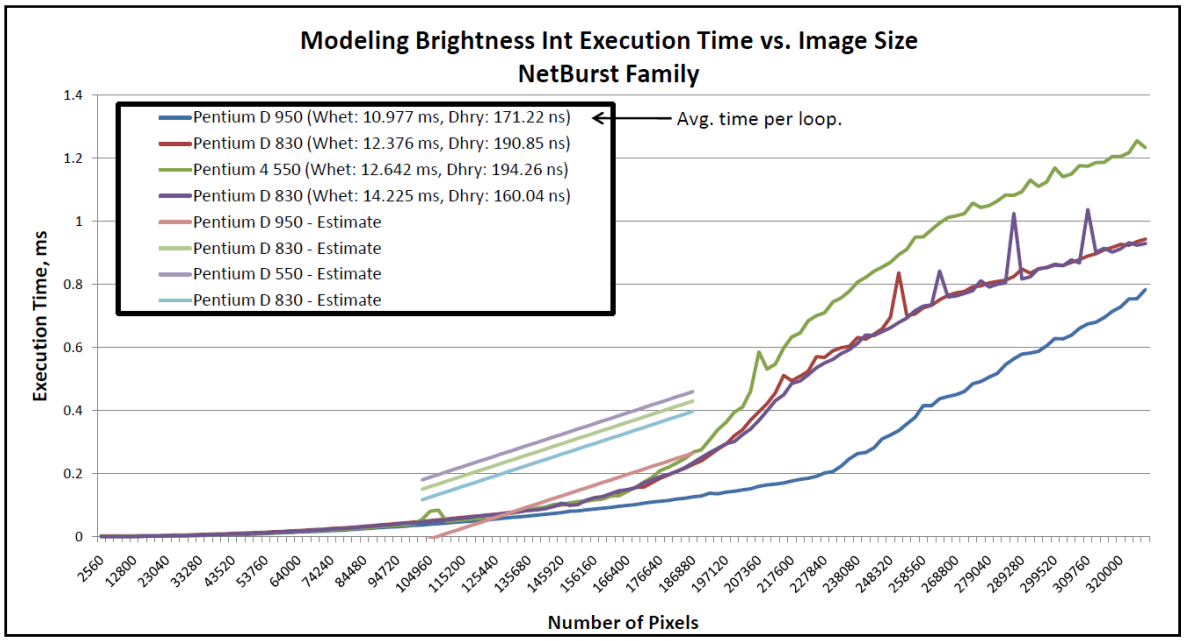

Figure 6. Brightness, Integer (NetBurst Family)

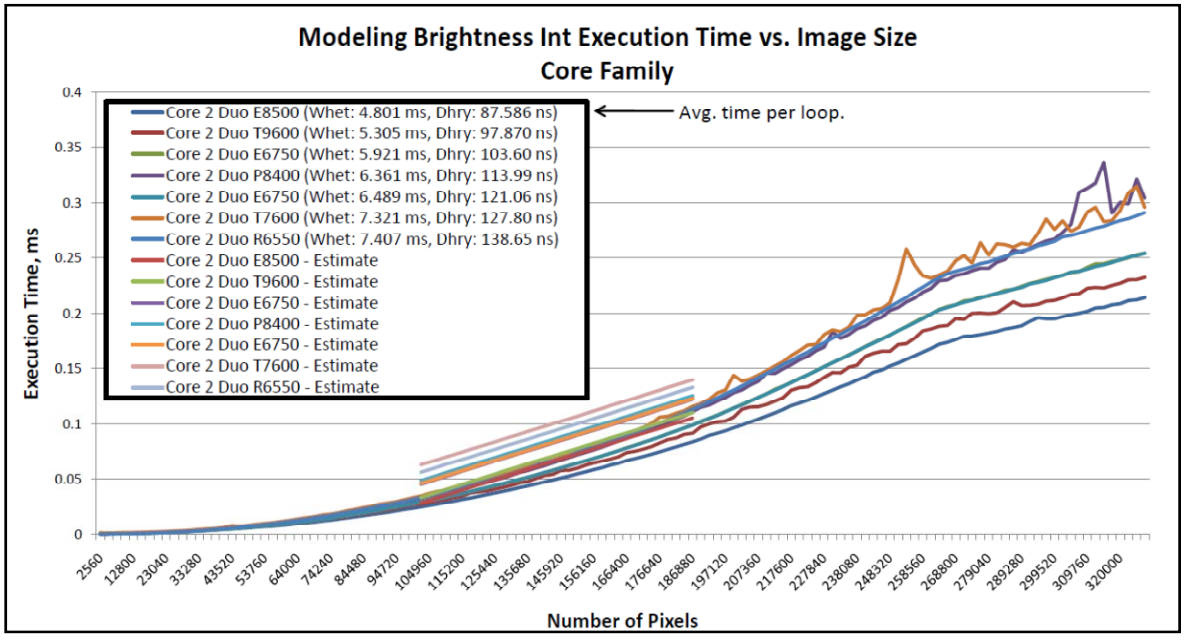

Figure 7. Brightness, Integer (Core Family)

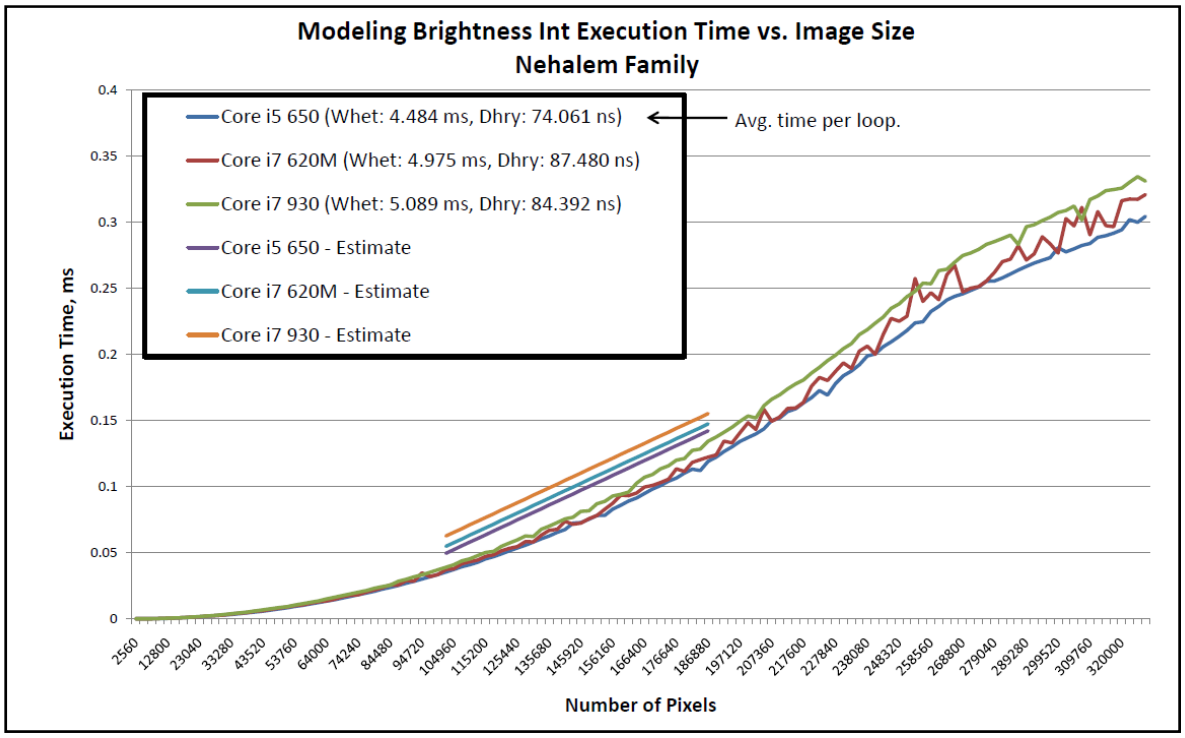

Figure 8. Brightness, Integer (Nehalem Family)

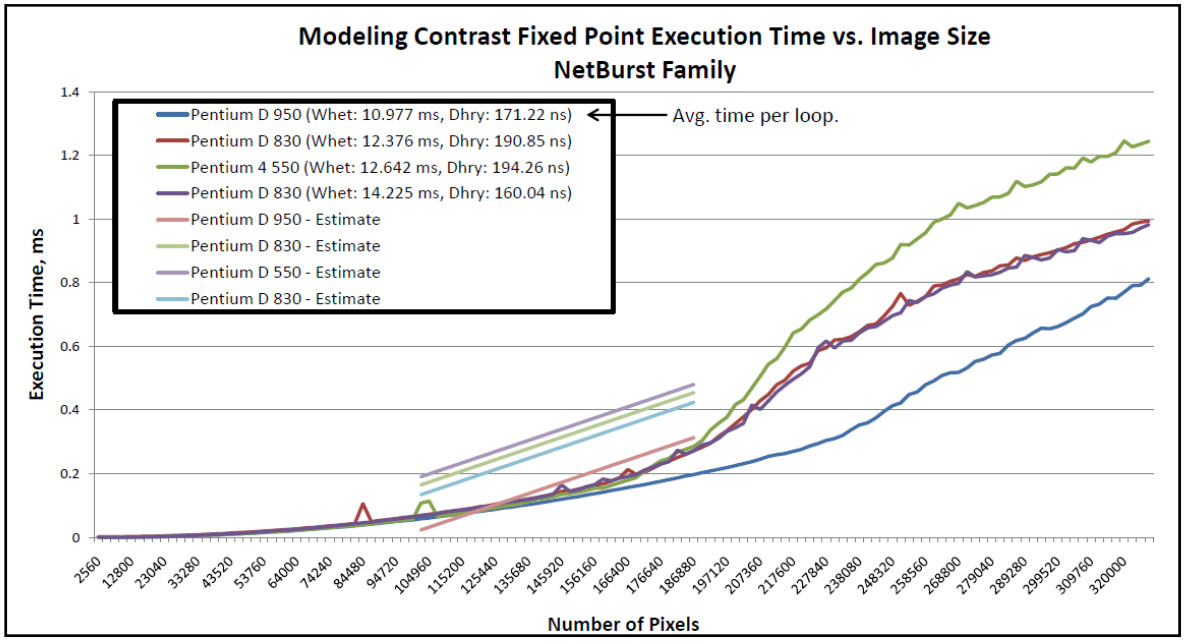

Figure 9. Contrast, Fixed Point (NetBurst Family)

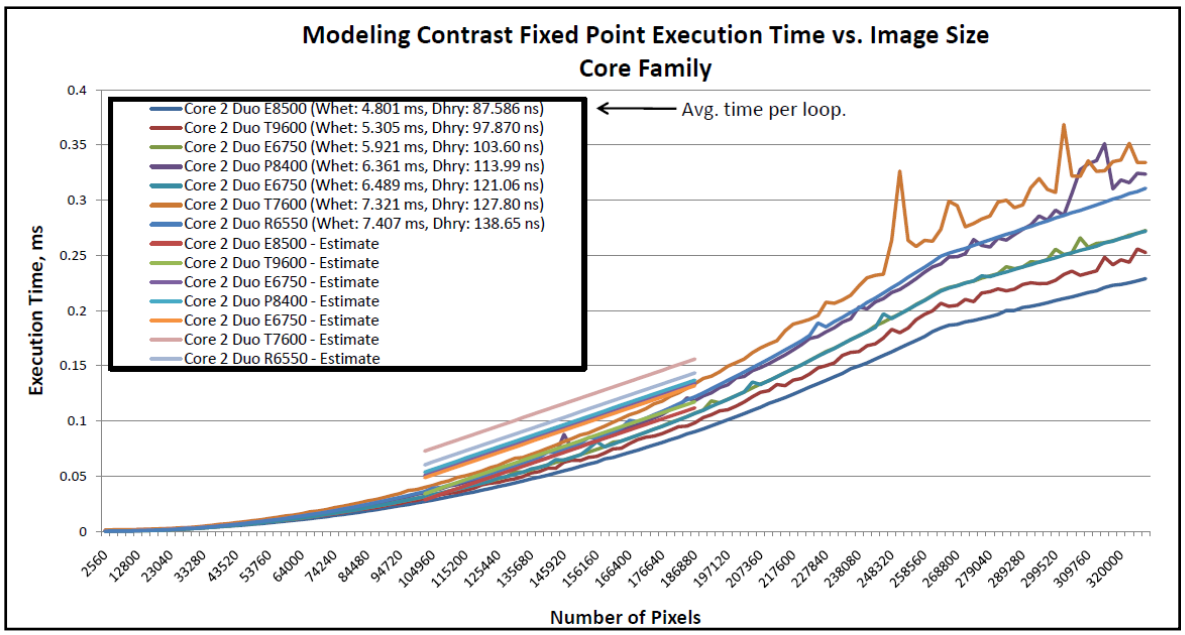

Figure 10. Contrast, Fixed Point (Core Family)

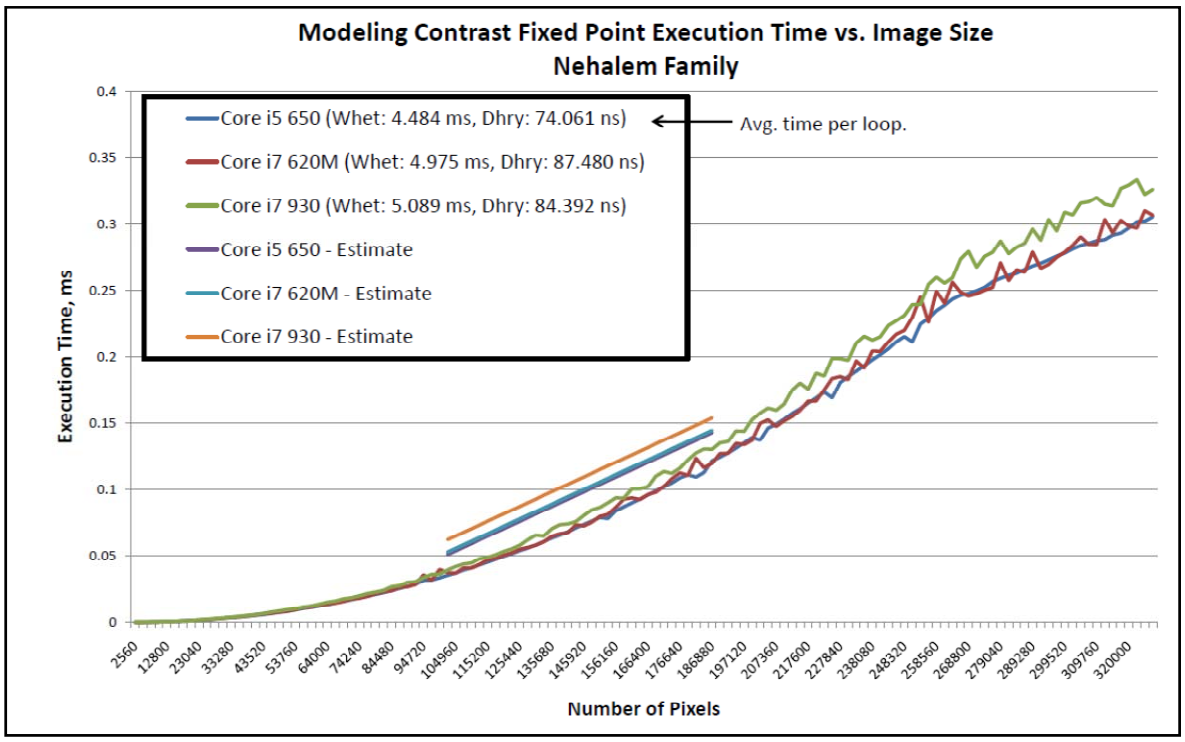

Figure 11. Contrast, Fixed Point (Nehalem Family)

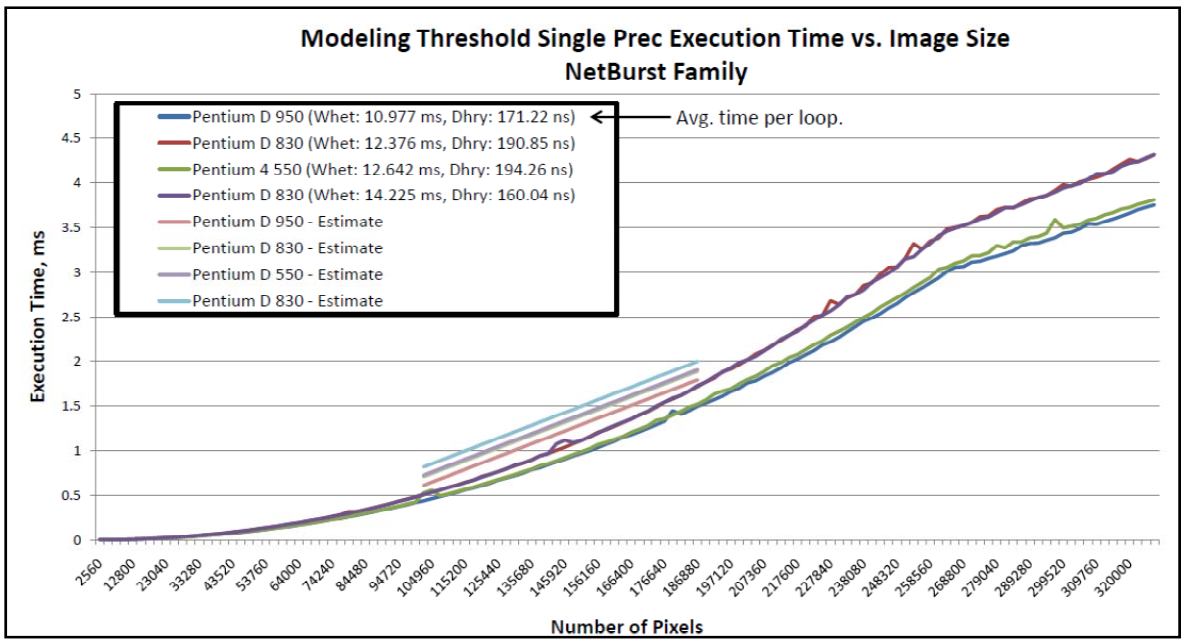

Figure 12. Threshold, Single Precision Floating Point (NetBurst Family)

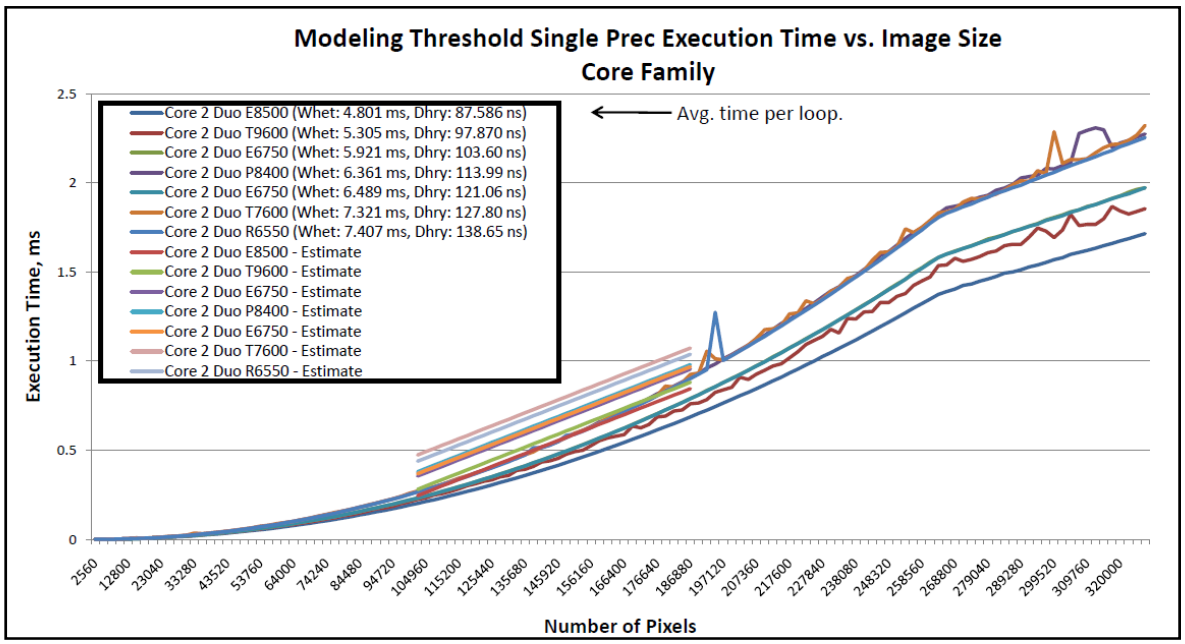

Figure 13. Threshold, Single Precision Floating Point (Core Family)

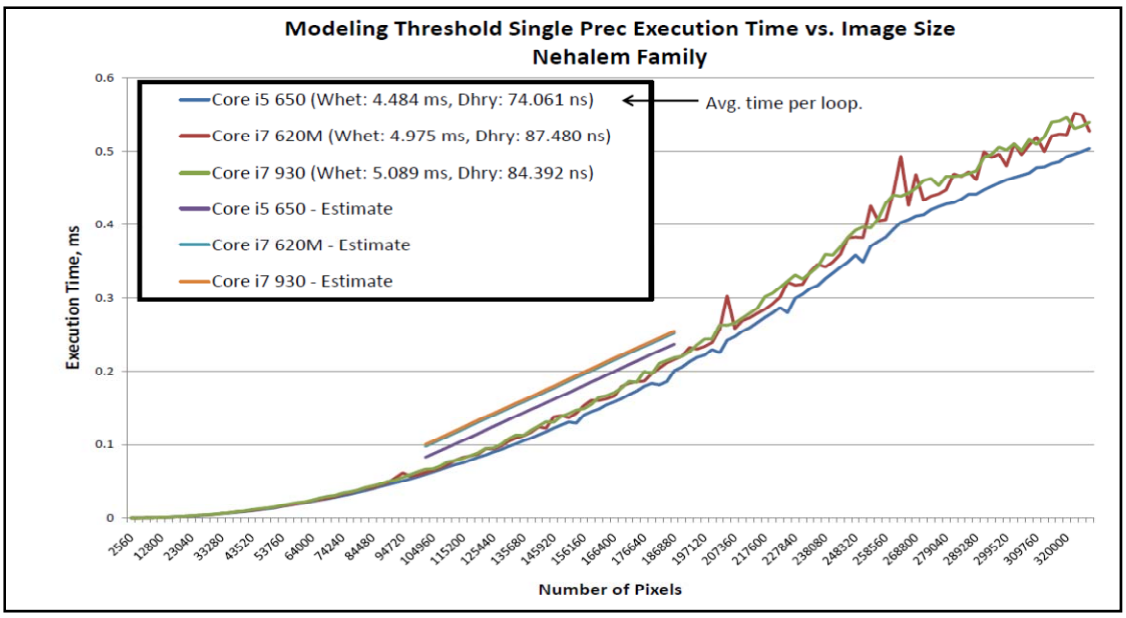

Figure 14. Threshold, Single Precision Floating Point (Nehalem Family)

The first objective is to determine a method for using the quantities available to draw conclusions about the relative performance of processors within a family.

The question is whether the execution times of the IP codes for different processors within a family monotonically increase with respect to Whetstone execution time, Dhrystone execution time, and the value of the estimation function.

To determine whether the execution time of an IP code is increasing monotonically with respect to another quantity, there must first be an ordering of IP code execution times for a group of processors. Consider Figure 6, for example. For the NetBurst family of processors, it is fairly clear that for larger image sizes, there is a clear ordering of processors in terms of performance (since the data for the Pentium D 830s showed similar results, the average of the points are taken along this line and considered a single case for the purposes of this discussion). For instance, the Pentium 4 550 shows the worst performance, the Pentium D 830 shows the next worst performance, and the Pentium D 950 performed the best. Based on this data, the performance of the Brightness Integer is not, strictly speaking, monotonically increasing with respect to Whetstone. The Pentium 4 550 performed the worst on the Brightness Integer; however, one of the Pentium D 830s performed the worst on Whetstone. Neither were the Brightness Integer results monotonically increasing with respect to Dhrystone, as the Pentium D 950 performed the best on the Brightness Integer, but the same Pentium D that showed the poorest Whetstone performance showed the best Dhrystone performance. However, assuming that these data points for the Pentium D 830 were anomalous, then the monotonic relationship does hold. The graph shows that the Brightness Integer results are monotonically increasing with respect to the estimation function (the estimation functions coalesced the two Pentium Ds into a single curve). However, this pertains only to only large image sizes. If the image were zoomed in on the left side of the graph, the data might exhibit different characteristics. The results in Figure 9 showing contrast fixed points for the NetBurst are similar to the results in Figure 6. Hence, similar conclusions can be drawn about those processors and their performance with these applications.

In the other graphs, other complexities arise. For example, while the data in Figures 6 and 9 can be easily divided into two sections for which at least one of those sections has a clear ordering of processors in terms of performance, other graphs cannot be so easily divided in this manner. Figure 7 shows how the Core 2 Duo P8400 changes in its ranking at least twice as image size increases. For medium size images, in general, the bottom three performing processors—starting with the worst—are the Core 2 Duo T7600, the Core 2 Duo R6550, and the Core 2 Duo P8400. However, for the largest image sizes, the P8400 drops below the performance of the other two for many image sizes, including the largest size. Hence, ordering relationships must be given with respect to specific image sizes. However, it is clear that the Brightness Integer performance is not monotonically increasing with respect to Whetstone or Dhrystone for any case on the right half of the graph. The R6550 performed the worst on Whetstone and Dhrystone; however, it did not perform the worst on the Brightness Integer for medium and large image sizes. The estimation function appears to be a better ordering for most image sizes, but it is not always correct. For example, for the largest image size, the P8400 performed the worst, whereas the estimation function reports that the T7600 was the worst. Several other graphs present this difficulty—to varying degrees—of changing rankings of processors as image size changes: Brightness Integer for Nehalem, Contrast Fixed Point for Core, Contrast Fixed Point for Nehalem, and Threshold for Nehalem. However, some of these cases could be remedied by the application of a noise reduction filter.

Another perspective on evaluating the estimation functions is their relative error. It is clear that because non-linear performance curves are being fit with linear functions, the error may vary widely. This is indeed the case for some image sizes. For large and small images, the error is generally larger. However, for the middle-size images, there are some cases for which the fit is quite good. Note that the accuracy of the fit also depends on which processor is considered. For example, the Core 2 Duo R6550 is a much flatter curve, especially for the Brightness Integer, and as a result it is a much better candidate for approximation with a linear function.

A point of interest in future work will be to attempt to fit the curves with non-linear or piece-wise functions. It is likely that a more accurate fit exists for a particular algorithm if it is determined that the algorithm is normally used only on a restricted range of image sizes. Another task for future work is to determine if these approaches will predict the performance of future processors. Although the approach shows promise within a family, its usefulness across families is an open question. The processors in the Core family generally outperformed the Nehalem. The current models do not have a method for taking into account how major differences between microarchitectural designs will affect performance.

VI. CONCLUSIONS AND RECOMMENDATIONS

Classic benchmarks (Dhrystones and Whetstones) are, in some cases, good processor speed comparison tools. In many cases, their results mapped well into the results of the real-world imaging algorithms that were discussed in this report. Higher Dhrystone and Whetstone results often mapped well into smaller execution times for IP codes. However, they are not good tools for estimating the execution time of real-world algorithms. Note that these benchmarks are valid comparison tools only if these guidelines are followed:

- The same compiler is used for each processor.

- The same optimization parameters are used during the compile process.

- If possible, we highly recommend using the same executable on each processor to be compared.

If good timing information is required for a given code, it is highly recommended that the execution time for that code be measured on the processor it will reside on. It is likely that Moore's Law will continue to provide designers with more transistors for new processor designs, and it is likely that this will result in more processors per chip. Programming these devices to get maximum throughput will require careful algorithm partitioning and control over thread-to-processor assignment. Balancing the per-processor load is not trivial and will require a talented and knowledgeable software team.

From classic benchmark (Dhrystone and Whetstone) data, the following was observed:

- The overall trend indicates that newer processors have larger ratings and are faster.

- In most (but not all) cases, the ratings appear to be correlated. In comparing two processors, if processor A has a higher Dhrystone rating than processor B, then usually the Whetstone ratings for processor A will also be higher than that of processor B. This is especially true for newer processors whose launch dates are the first quarter of 2007 and later.

- In comparing the performance of newer processors (launch dates that are in the first quarter of 2007 and later), it appears that Dhrystone performance has continued to improve with time, but Whetstone performance has improved at a slower rate.

- The AMD Athlon dual-core processor is a dual-boot machine where benchmarks were compiled under Linux (GNU compiler) and Microsoft Visual Studio Version 8. The results give an interesting view into the role of the compiler and/or operating system in use. The Float Whets for both are almost identical, but the integer performance is significantly higher (2699 versus 1979) for the Visual Studio 8 compilation run under Windows 7. However, the Dhrystone performance is significantly higher (6834 versus 3869) for the GNU compiler run under Linux.

From the IP benchmark data, the following was observed:

- Within a particular family, faster clock rates usually translate into a better performance.

- Processors designed for low power applications (for example, the Pentium M and Atom processors designed specifically for battery operated netbook and notebook computers) sacrifice computing performance to achieve their low power attributes.

- Modern (the third quarter of 2007 and newer) dual- and quad-core processors are capable of executing the full-frame (640-by-512 pixels), "simple" IP algorithms provided in real time (less than 2.0 milliseconds) on a single processor. Some of these processors (especially some in the Core 2 Duo family) dissipate power low enough to make them candidates for incorporation into military systems using 3U size cards.

- Modern dual- and quad-core processors appear capable of executing moderately complex full-frame IP algorithms given here in real time but will likely require the full resources of the processors.

- Modern quad-core processors appear capable of executing "very complex" full-frame IP algorithms given in real time but will likely require the full resources of all four cores. Given the electrical power required for these processors, this does not appear to be a good solution.

- Most modern processors appear to be quite capable of processing small subimage areas for the complex tasks given here. For example, the convolution of an 18-by-18 image using a 3-by-3 kernel can be done on the low power Core 2 Duo P8400 in 11.1 milliseconds on a single core. This implies that many complex tasks, such as tracking potential targets after they have been separated from clutter, can be accomplished in real time by modern processors.

- Floating point performance of many processors rivals that of integer and fixed point operations.

- For some (dual-core) processors, there were some experiments showing speedups nearly equal to the number of cores. However, in general this was not the case and we were not able to achieve a speedup near four for any of the quad-core processors.

Observations from the attempts to use classic benchmark data to predict IP execution times included in the following:

- Measured IP benchmark execution times and predicted (using classic benchmark data) IP benchmark execution times were often highly correlated for modern processors (those with a launch date of the first quarter of 2007 or earlier) but less so for older processors.

- Because of the nonlinear nature of some of the performance plots shown in Figures 6 through 14, it is likely that better fits could be obtained with a nonlinear curve fit.

Given the observed data, in general, it is not possible to reliably and accurately predict IP execution times with only Dhrystone and/or Whetstone performance data.

REFERENCES

1. Demme, J. and Sethumadhavan, S., "Approximate Graph Clustering for Program Characterization," ACM Transactions on Architecture and Code Optimization, Volume 8, New York, NY, January 2012.

2. Kainaga, M.; Yamada, K.; and Inayoshi, H., "Analysis of Spec Benchmark Programs," TRON Project Symposium Proceedings, pp. 208 –215, November 1991.

3. Munafo, R., "The SPEC Benchmarks," June 2012, http://mrob.com/pub/comp/benchmarks/spec.html

4. Aburto, A., "Geometric Mean or Median," June 2012, http://groups.google.com/group/comp.benchmarks/msg/728793aebd69097b?pli=1

5. Curnow, B. A. and Wichmann, H. J. "A Synthetic Benchmark," Computer Journal, Volume 19, pp. 43-49, 1976.

6. Weicker, R. P., "Dhrystone: A Synthetic Systems Programming Benchmark," Computing Practices, Volume 27, pp. 1013-1030, 1984.

7. Longbottom, Roy, "Roy Longbottom's PC Benchmark Collection," June 2012, http://homepage.virgin.net/roy.longbottom/index.htm

8. Frigo, Matteo, and Johnson, Steven G., FFTW.org Home Page, FFTW, June 2012, http://www.fftw.org/

9. "Measuring Processor Power," June 2012, http://www.intel.com/content/www/us/en/benchmarks/resources-xeon-measuring-processor-power-paper.html?wapkw=tdp

10. Wikipedia, "Cell (Microprocessor)," October 2011, http://en.wikipedia.org/wiki/Cell_(microprocessor)

LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

#	Number
AMD	Advanced Micro Devices
Avg	Average
CAD	Computer-Aided Design
CD	Compact Disk
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
Dhry	Dhrystone
DMIPS	Dhyrstone Millions Of Instruction Per Second
DLL	Dynamic Link Library
FFT	Fast Fourier Transform
Gbyte	gigabyte
GHz	gigahertz
IEEE	Institute of Electrical and Electronics Engineers
INT	Integer
IP	Image Processing
IR	Infrared
ISA	Instruction Set Architecture
Li	Linux
MIPS	Millions of Instructions Per Second
ms	millisecond
MWIPS	Millions of Whetstone Instructions Per Second
N/A	Not Applicable
nm	nanometer
ns	nanosecond
PC	Personal Computer
Q	Quarter
RAM	Random Access Memory
SPE	Synergistic Processing Element
TDP	Thermal Design Power
V	Vista

LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS (CONCLUDED)

VAX	Virtual Address eXtension
vs.	versus
W7	Windows 7
Whet	Whetstone

INITIAL DISTRIBUTION LIST

		Copies
Weapon Systems Technology Information Analysis Center Alion Science and Technology 201 Mill Street Rome, NY 13440	Ms. Gina Nash gnash@alionscience.com	Electronic
Defense Technical Information Center 8725 John J. Kingman Rd., Suite 0944 Fort Belvoir, VA 22060-6218	Mr. Jack L. Rike jrike@dtic.mil	Electronic
AMSAM-L	Ms. Anne C. Lanteigne hay.k.lanteigne.civ@mail.mil	Electronic
	Mr. Michael K. Gray michael.k.gray7.civ@mail.mil	Electronic
RDMR		Electronic
RDMR-CSI		Electronic
RDMR-WDG-C	Mr. Kenneth W. Pruitt kenneth.w.pruitt2.civ@mail.mil	Electronic
	Dr. Patrick A. La Fratta patrick.a.lafratta.civ@mail.mil	Electronic/Hardcopy